Interfacing Servo Motor with ATmega2560 and ARM7 LPC2148 (Fire Bird V)

Vishal H. Rajai and Joel M. Pinto
Under the mentorship of
Bhavin Upadhyay

July 10, 2014

e-Yantra Summer Internship – 2014
ERTS Lab
IIT Bombay

Contents

Abstract

This document is an introduction to Servo Motors and describes how to interface them using the Firebird V platform. It also serves as a report for the work done on servo motors by the authors during the Internship period.

0.1 Introduction

An RC servo consists of a dc motor, gear train, potentiometer, and some control circuitry all mounted compactly in a case. RC servos are commonly used in radio-controlled cars, airplanes, and boats to provide limited rotational motion to steer, move control surfaces, etc. RC servos are attractive for educational use in mechatronics, because they are relatively inexpensive and can put out about 42 oz/in of torque and can be controlled by a microcontroller (ATmega2560 or LPC2148 or any other controller).

Chapter 1

Servo Motors

1.1 Principle and Working

Servo system is a closed loop control system. Here instead of controlling a device by applying variable input signal, the device is controlled by a feedback signal generated by comparing output signal and reference input signal. After the device achieves its desired output, there will be no longer logical difference between reference input signal and reference output signal of the system. Then, third signal produced by comparing theses above said signals will not remain enough to operate the device further and to produce further output of the system until the next reference input signal or command signal is applied to the system. Hence the primary task of a servomechanism is to maintain the output of a system at the desired value.

As said in introduction, a servo has mainly dc motor, gear train, potentiometer, and control circuitry.

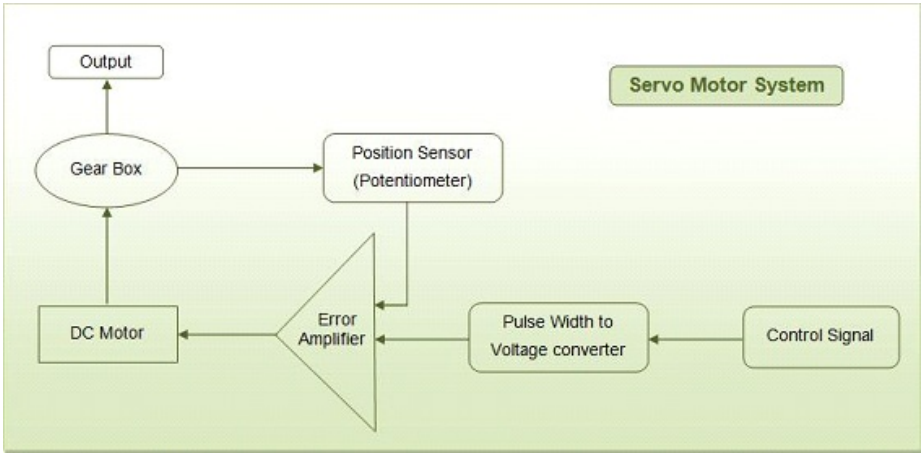

Figure 1.1: Closed loop control system model of Servo motor

Say at initial position of servo motor shaft, the position of the potentiometer knob is such that there is no electrical signal generated at the output port of the potentiometer. This output port is connected with one of the input

terminals of the error detector amplifier and electrical signal is given to rotate servo is given at another terminal of error detector amplifier. Now difference between signals on both terminals of the error detector will be amplified in the error detector amplifier which acts as the input power to the dc motor. Hence, the motor starts rotating in desired direction. As the motor shaft progresses the potentiometer knob also rotates as it is coupled with motor shaft with help of gear arrangement. As the angular position of the potentiometer knob progresses the output or feedback signal increases. After rotating motor shaft for desired angles, potentiometer knob reaches at such position that electrical signal generated in it are same as of external electrical signal given to error detector. At this condition, there will be no output signal from the amplifier to the motor input as there is no difference between external applied signal and the signal generated at potentiometer. As the input signal to the motor is nil at that position, the motor stops rotating and waits. This is how a simple conceptual servo motor works.

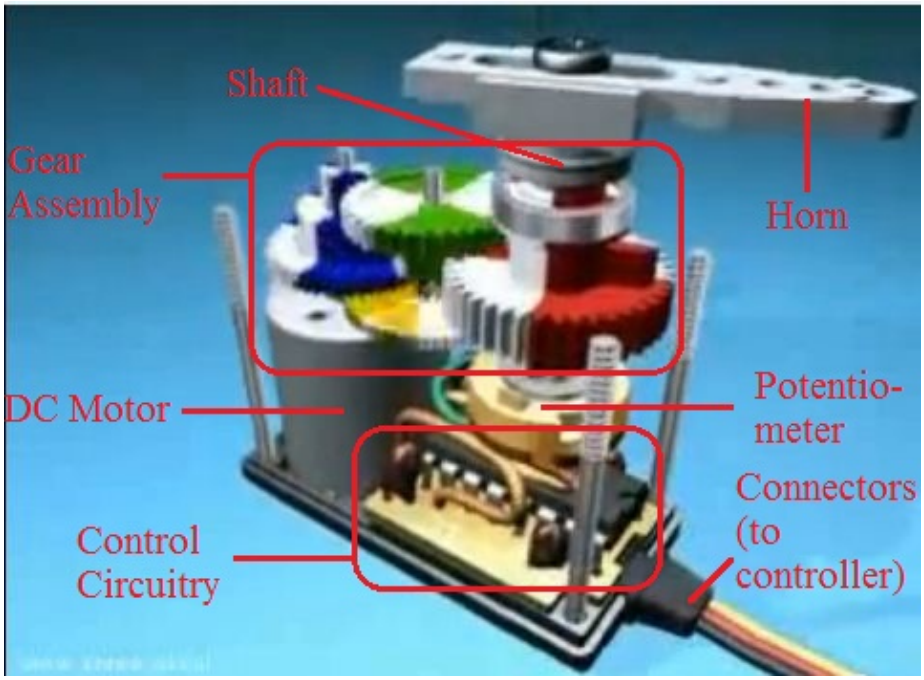

Figure 1.2: Internal structure and their labeling

1.2 Operating Servo Motor

Servo motor is operated using the wires provided in it. Out of the 3 wires, 2 of them (Red and Brown) are used to provide power and ground. The 3rd wire (yellow) is used to provide control signals to the Servo motor. Pulse Width Modulated (PWM) waves are used as control signals and the angular position is determined by the on-time of the pulse at the control input. The on-time period for the signal for rotating servo at particular angle depends on the servo used. Value of this time period is independent of the frequency of the signal given as PWM. Values for on-time for rotating servo back to 0°and to 180°are provided by the manufacturer in data sheet. Generally these values are around 1ms for 0°and around 2ms for 180°. Values for angles other than these can be easily calculated as they vary linearly with angles. For e.g. as in our case (NRS995), the on time for rotating servo to
$0°is0.6ms$
$180°is2.2ms$
Hence, the on time for rotating it to 90 °will be mid value of 0.6 and 2.2 as 90°falls in middle of 0°and 180°i.e. 1.4ms. Similarly, we get 1.0ms for 45°. Hence, the general formula for $T_{on}$for rotating servo by x°will be:

$T_{on} = (1.6/180)x° + 0.6$

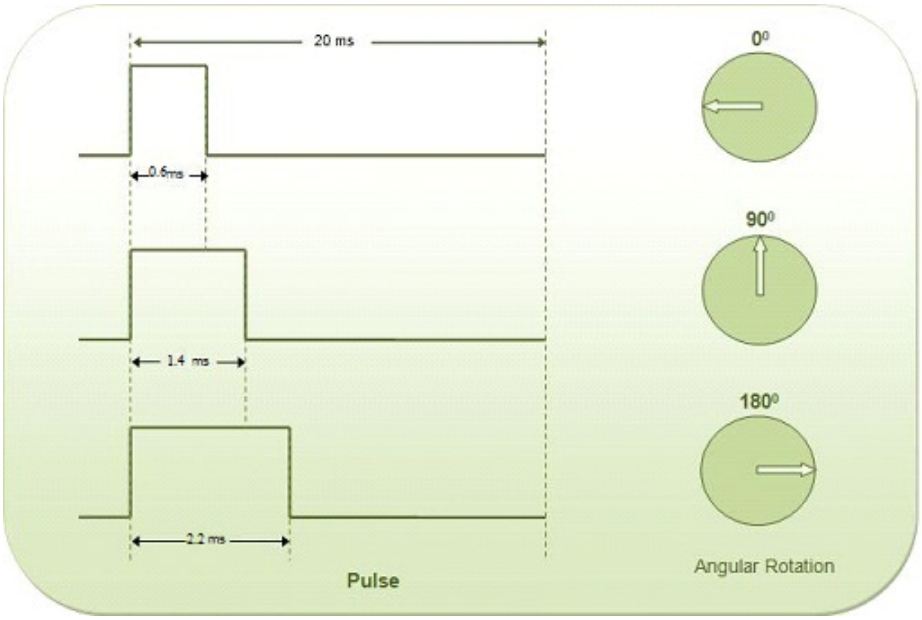

Figure 1.3: PWM signal

Generally, the operating range of frequency of PWM signal to servo is stated by datasheets to be 40 to 60 Hz. A sequence of such pulses is required to be passed to the servo to sustain a particular angular position. When the servo receives a pulse, it can retain the corresponding angular position for next time period. By experimentation, it was found that the servos internal controller is much more tolerant and worked well between 20Hz to 300Hz. No general variation of torque or speed was noticed with respect to changing the frequency

in this range. However, at very low frequencies, i.e., 5Hz to 15Hz, the servo gives a lot lesser torque than normal and the motion appears jerky. On the higher end of the frequency range, i.e., greater than 300 Hz, the servo starts heating up. We did not increase the frequency further as online forums suggest servos get damaged at such high frequencies.

1.3 Selection of a Servo

The typical specifications of servo motors are torque, speed, weight, dimensions, motor type and bearing type. The motor type can be of 3 poles or 5 poles. The pole refers to the permanent magnets that are attached with the electromagnets. 5 pole servos are better than 3 pole motor because they provide better torque. The servos are manufactured with different torque and speed ratings. A manufacturer may compromise torque over speed or speed over torque in different models. The weight and dimensions are directly proportional to the torque. Obviously, the servo having more torque will also have larger dimensions and weight. The selection of a servo can be made according to the torque and speed requirements of the application. The weight and dimension may also play a vital role in optimizing the selection such as when a servo is needed for making an RC airplane or helicopter or a robotic arm.

Chapter 2

AVR

2.1 Interfacing Servo with ATmega2560

Servo female connector used to connect a servo with a controller and their colour coding is shown in following figure 2.1

The power wire marked as no. 2 in figure is typically red. The ground wire is typically black or brown marked as no.1 in fig. and the signal pin is typically yellow or orange marked as no.3 in figure.

The position where this connector is connected to Fire Bird V to its corresponding power, control and ground pins is shown in fig 2.2

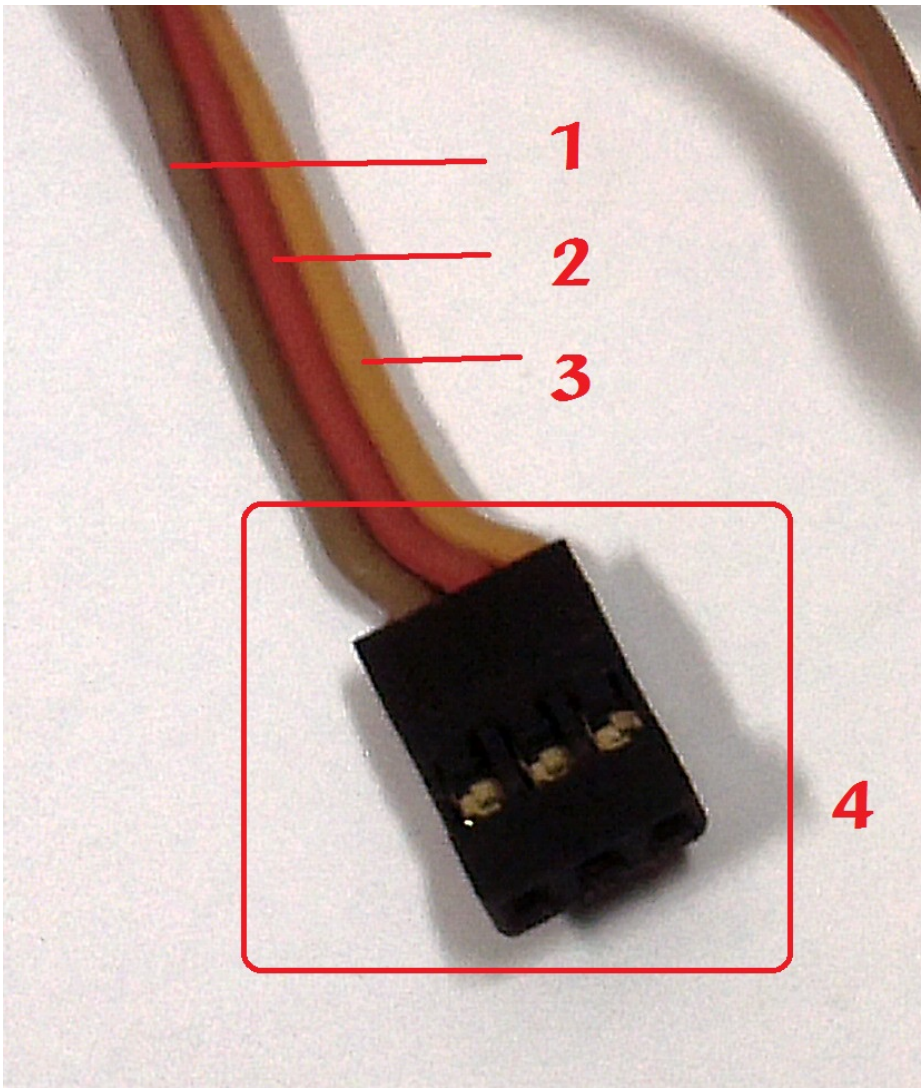

Figure 2.1: Servo female connector

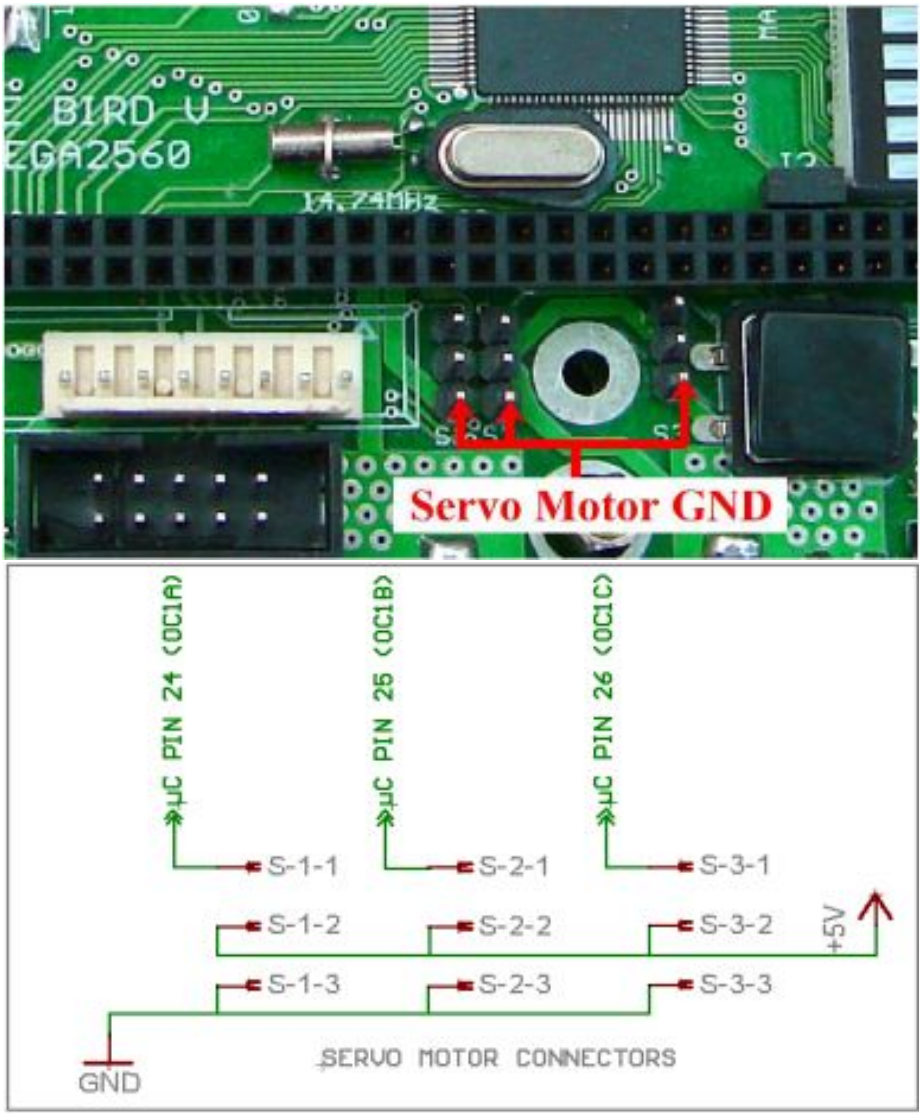

Figure 2.2: Servo motor connectors and its schematic in ATmega2560 Firebird V

2.2 Timers

There are in total 6 timers in ATmega2560

Two 8-bit Timers (Timer0, Timer2) and

Four 16-bit Timers (Timer1, Timer3, Timer4, Timer5)

8 bit timers have a resolution of 2 = 256 counts. Dividing the 20 ms pulse in 256 units, we obtain units of 78.125 μs. The resulting resolution we get is,

$Resolution = \frac{T_{on(180)} - T_{on(0)}}{unittime} = \frac{2.2ms - 0.6ms}{78.125\mu s}$

This means we have only 20 values between 0°and 180°!

For a more precise control of servo motor, we shall use the 16- bit Timer. Here, as we have only one servo to control, so we use Timer1. Rest all other 16-bit timers are similar as Timer1 and can be easily referred from ATmega2560 datasheet.

Timer1:

The Timer/Counter can be clocked internally, via the prescaler, or by an external clock source on the T_n pin. The clock source and edge the Timer/Counter uses to increment (or decrement) its value is selected by the Clock Select logic which is controlled by the Clock Select (CS12:0) bits located in the Timer/Counter control Register B (TCCR1B). The Timer/Counter is inactive when no clock source is selected. Output from clock select logic is referred to as the timer clock ($clkT_n$).

2.3 Registers

As said earlier, ATmega2560 has 12- 16 bit PWM channels with each 16 bit timer (Timer1, 3, 4, 5) having 3 channels: A, B and C. Here, we shall concentrate on registers required for generation of PWM signal which includes:

TCCR1A - Timer/Counter 1 Control Register A:

Bit	7	6	5	4	3	2	1	0	
(0x80)	COM1A1	COM1A0	COM1B1	COM1B0	COM1C1	COM1C0	WGM11	WGM10	TCCR1A
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 2.3: TCCR1A register contents

Here COM1x1:0 stands for Compare Output Mode for Timer1 in x channel.

Combined with the WGM13:2 bits found in the TCCR1B Register, these bits WGM11:0 in TCCR1A control the counting sequence of the counter, the source for maximum (TOP) counter value, and type of wave-form generation to be used. Modes of operation supported by the Timer/Counter unit are: Normal mode (counter), Clear Timer on Compare match (CTC) mode, and three types of Pulse Width Modulation (PWM) modes (Fast, Phase Correct, Phase and Frequency Correct mode).

Table below shows the mode of operation of Timer and the TOP value for different combinations of WGMn3:0 bits.

Mode	WGMn3	WGMn2 (CTCn)	WGMn1 (PWMn1)	WGMn0 (PWMn0)	Timer/Counter Mode of Operation	TOP	Update of OCRnx at	TOVn Flag Set on
0	0	0	0	0	Normal	0xFFFF	Immediate	MAX
1	0	0	0	1	PWM, Phase Correct, 8-bit	0x00FF	TOP	BOTTOM
2	0	0	1	0	PWM, Phase Correct, 9-bit	0x01FF	TOP	BOTTOM
3	0	0	1	1	PWM, Phase Correct, 10-bit	0x03FF	TOP	BOTTOM
4	0	1	0	0	CTC	OCRnA	Immediate	MAX
5	0	1	0	1	Fast PWM, 8-bit	0x00FF	BOTTOM	TOP
6	0	1	1	0	Fast PWM, 9-bit	0x01FF	BOTTOM	TOP
7	0	1	1	1	Fast PWM, 10-bit	0x03FF	BOTTOM	TOP
8	1	0	0	0	PWM, Phase and Frequency Correct	ICRn	BOTTOM	BOTTOM
9	1	0	0	1	PWM,Phase and Frequency Correct	OCRnA	BOTTOM	BOTTOM
10	1	0	1	0	PWM, Phase Correct	ICRn	TOP	BOTTOM
11	1	0	1	1	PWM, Phase Correct	OCRnA	TOP	BOTTOM
12	1	1	0	0	CTC	ICRn	Immediate	MAX
13	1	1	0	1	(Reserved)	–	–	–
14	1	1	1	0	Fast PWM	ICRn	BOTTOM	TOP
15	1	1	1	1	Fast PWM	OCRnA	BOTTOM	TOP

Figure 2.4: WGM bit description

Based on the different modes of operation of Timer, functionality of COM1x1:0 can be determined by following tables

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	Toggle OCnA/OCnB/OCnC on compare match.
1	0	Clear OCnA/OCnB/OCnC on compare match (set output to low level).
1	1	Set OCnA/OCnB/OCnC on compare match (set output to high level).

Figure 2.5: Compare Output Mode Non PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGM13:0 = 14 or 15: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match, set OCnA/OCnB/OCnC at BOTTOM (non-inverting mode).
1	1	Set OCnA/OCnB/OCnC on compare match, clear OCnA/OCnB/OCnC at BOTTOM (inverting mode).

Figure 2.6: Compare Output Mode Fast PWM

COMnA1 COMnB1 COMnC1	COMnA0 COMnB0 COMnC0	Description
0	0	Normal port operation, OCnA/OCnB/OCnC disconnected.
0	1	WGM13:0 =9 or 11: Toggle OC1A on Compare Match, OC1B and OC1C disconnected (normal port operation). For all other WGM1 settings, normal port operation, OC1A/OC1B/OC1C disconnected.
1	0	Clear OCnA/OCnB/OCnC on compare match when up-counting. Set OCnA/OCnB/OCnC on compare match when downcounting.
1	1	Set OCnA/OCnB/OCnC on compare match when up-counting. Clear OCnA/OCnB/OCnC on compare match when downcounting.

Figure 2.7: Compare Output Mode Phase Correct PWM and Phase and frequency Correct PWM

TCCR1B:

Bit	7	6	5	4	3	2	1	0
(0x81)	ICNC1	ICES1	–	WGM13	WGM12	CS12	CS11	CS10
Read/Write	R/W	R/W	R	R/W	R/W	R/W	R/W	R/W
Initial Value	0	0	0	0	0	0	0	0

Figure 2.8: TCCR1B bit description

Bit 7- ICNC1 is Input Capture Noise Canceler. As the name suggests, setting this bit enables noise canceler. This filters the input received from ICP1. This bit is generally set to 0 as we rarely use it.

Bit 6- ICES1 stands for Input Capture Edge Select. This bit controls which edge on ICP1 to use to trigger capture event. Clearing the ICES1 bit, a falling (negative) edge is used as trigger, and setting the ICES1 bit, a rising (positive) edge will trigger the capture.

When a capture is triggered, the counter value is copied into the Input Capture Register (ICR1). The event will also set the Input Capture Flag (ICF1), and this can be used to cause an Input Capture Interrupt, if this interrupt is enabled. When ICR1 is used as TOP value (decided on the basis of WGMn3:0 bits located in TCCR1A and TCCR1B Register), the ICP1 is disconnected and consequently the input capture function is disabled.

Bit 4: 3 : WGMn3:2: Waveform Generation Mode Function is similar to WGM discussed in TCCR1A

Bit 2:0 : CSn2:0: Clock Select The three clock select bits select the clock source to be used by the Timer/Counter

CSn2	CSn1	CSn0	Description
0	0	0	No clock source. (Timer/Counter stopped)
0	0	1	$clk_{I/O}$/1 (No prescaling
0	1	0	$clk_{I/O}$/8 (From prescaler)
0	1	1	$clk_{I/O}$/64 (From prescaler)
1	0	0	$clk_{I/O}$/256 (From prescaler)
1	0	1	$clk_{I/O}$/1024 (From prescaler)
1	1	0	External clock source on Tn pin. Clock on falling edge
1	1	1	External clock source on Tn pin. Clock on rising edge

Figure 2.9: Clock select bit description

TCNT1

Bit	7	6	5	4	3	2	1	0	
(0x85)	TCNT1[15:8]								TCNT1H
(0x84)	TCNT1[7:0]								TCNT1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 2.10: TCNT1 bit description

The two Timer/Counter I/O locations (TCNT1H and TCNT1L, combined TCNT1) give direct access, both for read and for write operations, to the Timer/Counter unit. To ensure that both the high and low bytes are read and written simultaneously when the CPU accesses these registers, the access is performed using an 8-bit temporary High Byte Register (TEMP). This temporary register is shared by all the other 16-bit registers..

Modifying the counter (TCNT1) while the counter is running introduces a risk of missing a compare match between TCNT1 and one of the OCR1x Registers.

Writing to the TCNT1 Register blocks (removes) the compare match on the following timer clock for all compare units.

OCR1A

Bit	7	6	5	4	3	2	1	0	
(0x89)	OCR1A[15:8]								OCR1AH
(0x88)	OCR1A[7:0]								OCR1AL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 2.11: OCR1A bit description

OCR1B

Bit	7	6	5	4	3	2	1	0	
(0x8B)	OCR1B[15:8]								OCR1BH
(0x8A)	OCR1B[7:0]								OCR1BL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 2.12: OCR1B bit description

OCR1C

Bit	7	6	5	4	3	2	1	0	
(0x8D)	OCR1C[15:8]								OCR1CH
(0x8C)	OCR1C[7:0]								OCR1CL
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 2.13: OCR1C bit description

OCR1A/B/C is Output Compare Registers of 16-bit size. Its value is continuously compared with the counter value (TCNT1). A match can be used to generate an Output Compare interrupt, or to generate a waveform output on the OC1x pin.

To ensure that both the high and low bytes are written simultaneously when the CPU writes to these registers, an 8-bit temporary register, High Byte Register (TEMP) is used. This temporary register is shared by all the other 16-bit registers.

ICR1

Bit	7	6	5	4	3	2	1	0	
(0x87)	ICR1[15:8]								ICR1H
(0x86)	ICR1[7:0]								ICR1L
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 2.14: ICR1 bit description

ICR1- Input Capture Register1 is updated with the counter (TCNT1) value each time an event occurs on the ICP1 pin (or optionally on the Analog Comparator output for Timer/Counter1). The Input Capture can be used for defining the counter TOP value.

TIMSK1

Bit	7	6	5	4	3	2	1	0	
(0x6F)	–	–	ICIE1	–	OCIE1C	OCIE1B	OCIE1A	TOIE1	TIMSK1
Read/Write	R	R	R/W	R	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

Figure 2.15: TIMSK1 bit description

Bit 5 : ICIE1 Timer/Counter1, Input Capture Interrupt Enable When this bit and the I-flag in the Status Register are set (interrupts globally enabled), the Timer/Counter1 Input Capture interrupt is enabled.

Bit 3/2/1 : OCIE1C/B/A Timer/Counter1, Output Compare C/B/A Match Interrupt Enable When this bit and the I-flag in the Status Register are set (interrupts globally enabled), the Timer/Counter1 Output Compare C/B/A Match interrupt is enabled.

Bit 0 : TOIE1 Timer/Counter1, Overflow Interrupt Enable When this bit and the I-flag in the Status Register are set (interrupts globally enabled), the Timer/Counter1 Overflow interrupt is enabled.

2.4 Hardware method for Generating PWM

2.4.1 Generating PWM signal using Timer

PWM signals can be generated in 2 ways on ATmega2560:

1) Hardware method

2) Software method.

Looking in datasheet of a controller, one can get the number of PWM channels available for generating PWM signals. As for ATmega2560, it has 12-16 bit resolution PWM channels.

Here is a simplified view of the Timers when they are used for PWM:

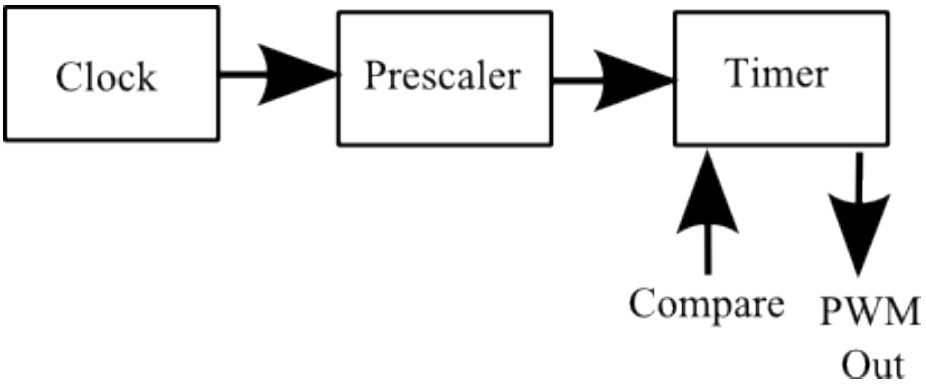

Figure 2.16: Block diagram of timer whem used as PWM

The clock

This is either the speed of any external crystal used or the internal clock speed of microcontroller. While compiling program we (in AVR lib) have to set the F_CPU variable to this value. There is only one clock speed per microcontroller.

The prescaler

The purpose of the prescaler is to divide the clock frequency by a given value so as to slow down the counting process in the timer. This slow-down factor is always a power of 2 like 1, 8, 32, 64, 128, 256, and 1024. Note that each Timer may only provide a subset of these prescaler values. Formula to work out how often the counter is incremented is:

$\frac{Prescaler}{clockfrequency}$

For e.g. as in ATmega2560, the clock speed is 14.7456 MHz

So, if prescaler is 64 than the counter will add one to its value every:

$64/14745600 \Rightarrow every 4.3402 \mu s$

Looking at it other way round, if the prescaler is 64 than the counter will add 230400 values to its present value in every 1 sec (14575600/64).

Note that if one timer provides multiple PWM channels then they share the same prescaler setting.

The prescaler is very helpful when the clock rate changes by a power of 2. i.e. changing the clock speed from 1 Mhz to 8 Mhz can easily be compensated by changing the prescaler from x to 8x without changing other code.

2.4.2 Modes of Operation

Most PWM timers can support the following modes:
1) Fast PWM mode,
2) Phase Correct PWM mode and
3) Phase and Frequency Correct PWM mode.

When using a PWM timer to drive only one thing then these different modes are pretty much of the same. But if we use them to drive multiple things (especially any kind of motor: DC motor or servo) from the same PWM timer then there is a subtle difference between them. Since all PWM channels, via the same timer, share the same waveform generator then the only thing that differentiates each channel is the 'comparator' value.

Fast PWM:
It uses the sawtooth waveform where the timer counter TCNT1, counts from BOTTOM to TOP and then it is simply allowed to overflow (or cleared at a compare match) to BOTTOM. Hence, we can provide a reference count value in register (OCR1x), so that whenever the counter has counts values above that reference value, the output at certain pin is high, else it is low. This is called the Inverted mode of operation.
Below is the list of all three modes in fast PWM:

- Inverted Mode - In this mode, if the counter value is greater than the reference value, then the output is set high, or else the output is low. This is represented in figure A above.

- Non-Inverted Mode - In this mode, the output is high whenever the reference value is greater than the current value in the counter and low otherwise. This is represented in figure B above.

- Toggle Mode - In this mode, the output toggles whenever there is a compare match. If the output is high, it becomes low, and vice-versa.

In fast PWM mode the counter is incremented until the counter value matches either one of the fixed values 0x00FF, 0x01FF, or 0x03FF (WGMn3:0 = 5, 6, or 7), the value in ICR1 (WGMn3:0 =14), or the value in OCR1A (WGMn3:0 = 15). The counter is then cleared at the following timer clock cycle.

Using ICR1 Register for defining TOP value works well when using fixed TOP value. By using ICR1, the OCR1A Register is free to be used for generating a PWM output on OC1A.
In fast PWM mode, setting the COM1x allows us to generate non-inverted PWM or inverted PWM as shown in table of COM1x discussed earlier. The actual OC1x value will only be visible on the port pin if the data direction for the port pin is set as output (DDR_OC1x).
The PWM waveform is generated by setting (or clearing) the OC1x Register at the compare match between OCR1x and TCNT1, and clearing (or setting) the OC1x Register at the timer clock cycle the counter is cleared (changes from TOP to BOTTOM).

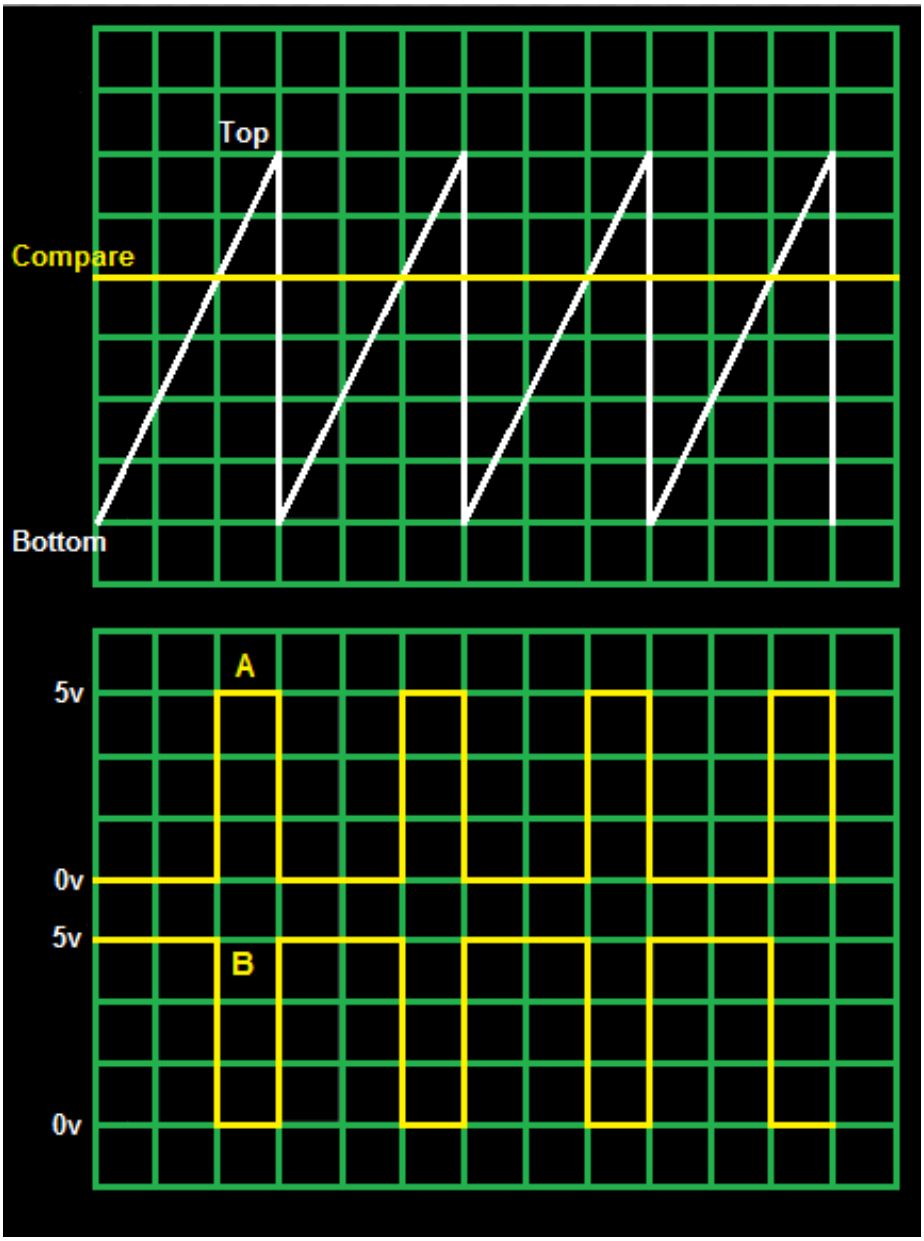

Figure 2.17: PWM signal generation Fast PWM

The PWM frequency for the output can be calculated by the following equation:
$f_{PWM} = \frac{f_{CPU}}{N*(1+TOP)}$
N = prescaler divider (1, 8, 64, 256, or 1024).
f_{CPU} =frequency of the crystal (=14745600 Hz for ATmega2560)

Value for OCR1x can be found using the following eq.:
$OCR1A = T_{on} \frac{f_{CPU}}{N}$
T_{on}= on-time period for rotating servo at particular degree
For servo motor NRS995 it is,
0.6 ms $\Rightarrow 0^\circ$
1.4 ms $\Rightarrow 90^\circ$
2.2 ms $\Rightarrow 180^\circ$

Phase and Frequency Correct PWM and Phase Correct PWM: There is no difference between both of these modes if we are not changing the value of TOP on the fly. The major difference is that fast PWM mode counted repeatedly from BOTTOM to TOP to generate a sawtooth waveform whereas these any phase correct modes will count up from BOTTOM to TOP, and then from TOP to BOTTOM so rather than a sawtooth they generate a triangular waveform:

So the first thing to note with these modes is that frequency is now halved. With fast PWM the frequency was relative to TOP but now it is 2(TOP) because we are both counting up to TOP and then down to BOTTOM.

As it can be seen from the figure that red and blue pulses are no longer aligned at the end of the pulse as they were for fast PWM but they are now centered around the TOP value. This will minimize the positional errors as in fast PWM.

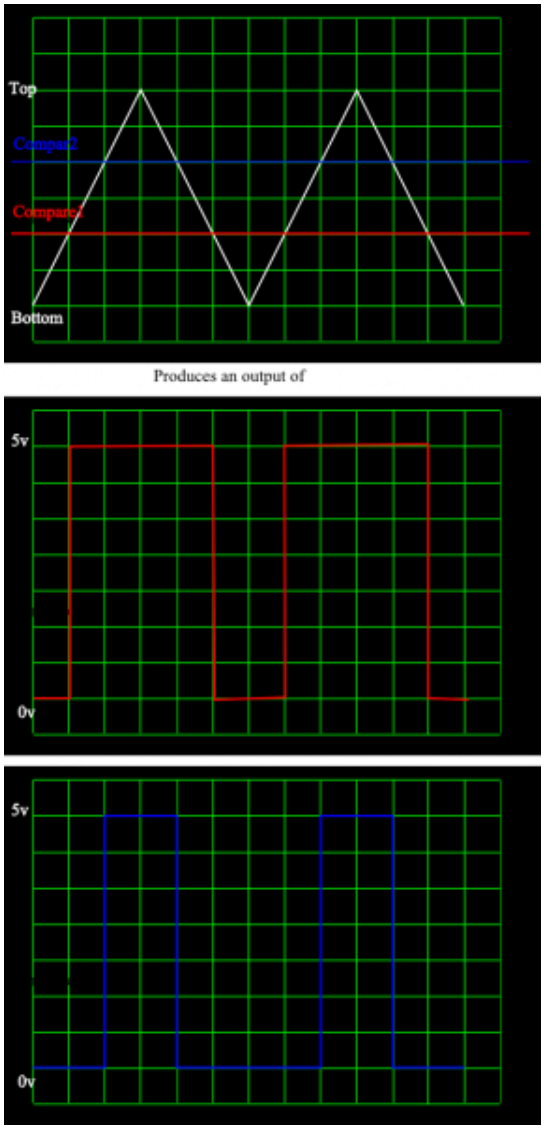

Figure 2.18: PWM signal generation Phase Correct

2.5 Generating PWM in software

Requirement of generation PWM in software

The delay method is fine if we are only using a few servos. But while trying to drive 10 servos then each of those delays adds up. So if each servo is sent a position pulse and then pausing for 20ms then with 10 servos we would be pausing for a total of 200ms every time around the main loop. Whilst the pauses, rest of our code cannot be doing anything. So each servo actually ends up getting the full delay of 200ms and will therefore be very jerky. This also means that we would be reading our sensors every 200ms. In other words, the more servos are added then the slower, and jerkier, the whole robot becomes.

So to solve this issue we need to get rid of all of those delay loops so that the main program can run at full speed and so that each servo is 'refreshed' every 20ms no matter how many additional servos you connect.

The solution is to use software interrupts to simulate PWM which, in theory, could be used to produce PWM on any general output pin on your microcontroller and thus provide a much larger number of PWM output channels.

The big caveat is that this will never be quite as precise as doing it using hardware because it is more susceptible to being interrupted by other interrupt service routines. However, if the devices that are being driven are not time critical then this gives an acceptable solution.

The basic technique is that each software PWM channel requires:
1) Turn on the output pin for the PWM channel and

2) queue up a toggle event that will be run at a later date

3) When the time has elapsed you change the output pin to low and

4) Queue up a toggle event that will be run at a later date.

When the time has elapsed go to step 1 The sum of the two delays will dictate your PWM frequency, and the duration of the delay in step 2 will dictate the duty cycle. If we can perform all of this under interrupts then your main program can run without pauses and each servo should be refreshed every 20ms.

2.6 Selection of Timer

Servos generally need a signal every 20ms and the high pulse should be between 1ms and 2ms. If our repeating frequency is every 20ms then an 8 bit timer (having 256 increments) would mean each increment would represent 20ms/256 or 0.078ms. However, most of the 20ms is dormant and all we have to play with is the pulse duration between 1ms and 2ms and we can only break this down into 1/0.078 or 12.8 steps. So an 8 bit timer would only give us 12 independent positions for a normal servo. Most servos (depending on the manufacturer and model) have around 90 possible steps - so only being able to access 12 of them isn't making the most of the servo - although it may be enough for particular purpose. Using a 16 bit timer would give a timer granularity of 20ms/65536 or 0.0003ms and so for the 1ms to 2ms pulse width would give 3,276 individual steps. So, we can make the most of the abilities of the servo. Also, the 16 bit Timers allows us to precisely set the value of TOP in order to generate an exact 20ms signal.

2.7 Code

After learning about PWM, registers, etc. we shall now program controller ATmega2560 to operate servo motor.

Let us use 16bit Timer1 in Fast PWM Non-Inverting mode (with fixed frequency).

From WGM bit description table, ICR1 fixes the TOP (i.e. TOP = ICR1) and OCR1A decides the Pulse Width.

To run the servo we must fix the PWM frequency around 40-60 Hz. Let us chose 50Hz (20ms Time Period)

With fixed value of $f_PWM = 50Hz$, $f_CPU = 14.7456MHz$ and $N = 256$, value of TOP and hence the ICR1 can be calculated from the eq. of frequency of PWM discussed above as,

$TOP = \frac{f_{CPU}}{(f_{PWM})N} - 1$

Hence, $ICR1 = TOP = 1023$

Since Timer is running on Non-Inverting mode, 20ms = 1152 counts of the timer Hence,

$0.6ms \Rightarrow 34.56 counts of the timer \Rightarrow 0°$

$2.2ms \Rightarrow 126.72\ counts of the timer \Rightarrow 180°$

Code:

```
#define F_CPU 14745600
#include <avr/io.h>
#include <avr/interrupt.h>
#include <util/delay.h>

void port_init()
{
DDRB = DDRB | 0x20; //making PORTB 5 pin output
PORTB = PORTB | 0x20; //setting PORTB 5 pin to logic 1
}

void timer1_init()
{
TCCR1A = 0x00;

ICR1 = 1023; //TOP = 1023
TCNT1H = 0xFC; //Counter high value to which OCR1xH value is to be compared with
TCNT1L = 0x01; //Counter low value to which OCR1xH value is to be compared with
OCR1A = 1023;
TCCR1A = 0xAB;
//COM1A1=1, COM1A0=0; COM1B1=1, COM1B0=0; COM1C1=1 COM1C0=0
//For Overriding normal port functionality to OCRnA outputs. WGM11=1, WGM10=1. Along
//WGM12 in TCCR1B for Selecting FAST PWM Mode

TCCR1B = 0x0C; //WGM12=1; CS12=1, CS11=0, CS10=0 (Prescaler=256)
}

void servo_1(unsigned char degrees)
```

```
{
float regval = ((float)degrees * 0.512) + 34.56;
OCR1A = (uint16_t) regval;
}

void servo_1_free (void)
{
OCR1A = 1023;
}

void init()
{
cli();
port_init();
timer1_init();
sei();
}

int main(void)
{
init();
servo_1(0);
_delay_ms(1000);
servo_1(90);
_delay_ms(1000);
servo_1(180);
_delay_ms(1000);

while(1)
{

}
return 0;

}
```

2.8 Other Examples

Other things that we tried on servo motor include:

Rotating motor with 1°precision,

Speed control of Servo motor,

Practically finding operating frequency range of a servo motor.

An API is also designed which can control up to 24 servo motors simultaneously with speed control for each motor.

Comments in the code include description of the algorithm and its explanation.

Chapter 3

ARM

3.1 Interfacing Servo with ARM7 LPC2148

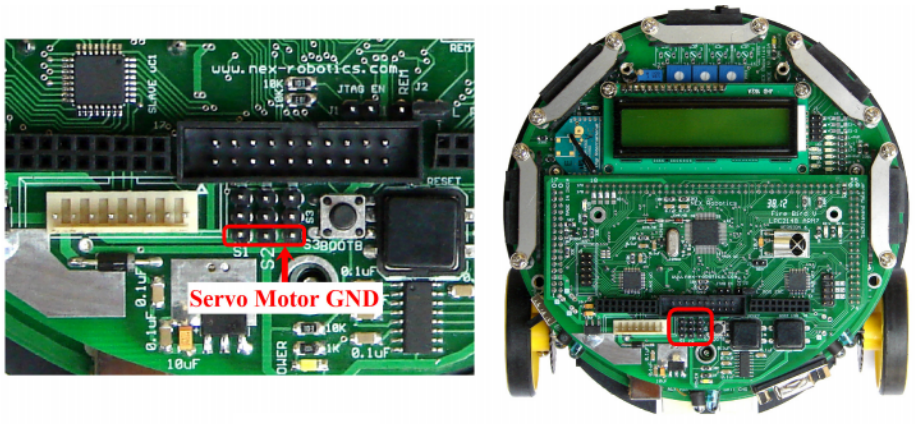

Figure 3.1: Servo connectors location and schematic

Location of servo connectors and its schematic are shown in the above figure:

3.2 PWM Programming in LPC2148

In LPC2148 we have 7 match registers inside the PWM block. Generally the first Match register PWMMR0 is used to generate PWM period and hence we are left with 6 Match Registers PWMMR1 to PWMMR6 to generate 6 Single Edge PWM signals. Double edge PWM uses 2 match registers hence we can get only 3 double edge outputs. Outputs of the PWM are delivered to actual pins on the controller unit. Table below shows the output pin no. for corresponding PWM output.

PWM1 output corresponds to PWMMR1 (PWM Match Register 1), PWM2 output corresponds to PWMMR2, and so on. Also the PWM function must be selected for the PINs mentioned above using appropriate PINSEL registers (PINSEL0 for PWM1, 2, 3, 4, 6 and PINSEL1 for PWM5). These registers are discussed in the following section.

LPC2148 supports 2 types of PWM:

1) Single Edge PWM : Pulse starts with new Period i.e. pulse is always at the beginning

2) Double Edge PWM : Pulse can be anywhere within the Period. Rules for Single Edged PWM are as:

1) All single edged PWM outputs will go high at the beginning of a PWM cycle unless their match value is 0.

2) Each PWM output will go low when its match value is reached. If no match occurs i.e. Match value is greater than Period then the output will remain high.

Rules for double edge controlled PWM outputs:

1) The match values for the next PWM cycle are used at the end of a PWM cycle (a time point which is coincident with the beginning of the next PWM cycle), except as noted in rule 3.

2) Match value equal to 0 or the current PWM rate (the same as the Match channel 0 value) have the same effect, except as noted in rule 3. For example, a request for a falling edge at the beginning of the PWM cycle has the same effect as a request for a falling edge at the en d of a PWM cycle.

3) When match values are changing, if one of the "old" match values is equal to the PWM rate, it is used again once if neither of the new match values are equal to 0 or the PWM rate, and there was no old match value equal to 0.

4) If both a set and a clear of a PWM output are requested at the same time, clear takes precedence. This can occur when the set a d clear match values are the same as in or when the set or clear value equals 0 and the other value equals the PWM rate.

5) If a match value is out of range (i.e. greater than the PWM rate value), no match event occurs and that match channel has no effect on the output. This means that the PWM output will remain always in one state, allowing always low, always high, or "no change" outputs.

Note that for Single Edged PWM, these Pins PWMMR1-6 are set to High by default when a new Period starts i.e. when TC is reset as per the PWM Rules given above.

3.3 PWM Register

1) PWMTCR: PWM Timer Control Register

This register is used to control the Timer Counter inside the PWM block. Only Bits: 0, 1 and 3 are used rest are reserved.

– Bit 0: This bit is used to Enable/Disable Counting. When 1 both PWM Timer counter and PWM Prescale counter are enabled. When 0 both are disabled.

– Bit 1: This bit is used to reset both Timer and Prescale counter inside the PWM block. When set to 1 it will reset both of them (at next edge of PCLK).

– Bit 3: This is used to enable the PWM mode i.e. the PWM outputs.

Other Bits: Reserved.

2) PWMPR: PWM Prescale Register

PWMPR is used to control the resolution of the PWM outputs. The Timer Counter (TC) will increment every PWMPR+1 Peripheral Clock Cycles (PCLK).

3) PWMMR0 to PWMMR6: Match Registers

These are the seven Match registers as explained above which contain Pulse Width Values i.e. the Number of PWMTC Ticks.

4) PWMMCR: PWM Match Control Registers

The PWM Match Control Register is used to specify what operations can be done when the value in a particular Match register equals the value in TC. For each Match Register we have 3 options: Either generate an Interrupt, or Reset the TC, or Stop which stops the counters and disables PWM. Hence this register is divided into group of 3 bits. The first 3 bits are for Match Register 0 i.e. PWMMR0, next 3 for PWMMR1, and so on:

1) Bit 0: Interrupt on PWMMR0 Match : If set to 1 then it will generate an Interrupt else disable if set to 0.

2) Bit 1: Reset on PWMMR0 Match : If set to 1 it will reset the Timer Counter i.e. PWMTC else disabled if set to 0.

3) Bit 2: Stop on PWMMR0 Match : If this bit is set 1 then both PWMTC and PWMPC will be stopped and will also make Bit 0 in PWMTCR to 0 which in turn will disable the Counters.

Similarly Bits 3,4,5 for PWMMR1 , Bits 6,7,8 for PWMMR2 , Bits 9,10,11 for PWMMR3 ,Bits 12,13,14 for PWMMR4 ,Bits 15,16,17 for PWMMR5 , Bits 18,19,20 for PWMMR6.

5) PWMIR: PWM Interrupt Register

If an interrupt is generated by any of the Match Register then the corresponding bit in PWMIR will be set high. Writing a 1 to the corresponding location will clear that interrupt.

Here:

1) Bits 0,1,2,3 are for PWMMR0, PWMMR1, PWMMR2, PWMMR3 respectively and

2) Bits 8,9,10 are for PWMMR4, PWMMR5, and PWMMR6 respectively. Other bits are reserved.

6) PWMLER: Latch Enable Register

The PWM Latch Enable Register is used to control the way Match Registers are updated when PWM generation is active. When PWM mode is active and we apply new values to the Match Registers the new values would not get applied immediately. Instead what happens is that the value is written to a Shadow Register it can be thought of as a duplicate Match Register. Each

Match Register has a corresponding Shadow Register. The value in this Shadow Register is transferred to the actual Match Register when:

1) PWMTC is reset (i.e. at the beginning of the next period)

2) And the corresponding Bit in PWMLER is 1.

Hence only when these 2 conditions are satisfied the value is copied to Match Register. Bit x in PWMLER corresponds to match Register x. I.e. Bit 0 is for PWMMR0, Bit 1 for PWMMR1,.. and so on. Using PWMLER will be covered in the examples section.

7) PWMPCR: PWM Control Register

This register is used for Selecting between Single Edged and Double Edged outputs and also to Enable/Disable the 6 PWM outputs which go to their corresponding Pins.

1) Bits 2 to 6 are used to select between Single or Double Edge mode for PWM 2, 3, 4, 5, 6 outputs.

1) Bit 2: If set to 1 then PWM2 (i.e. the one corresponding to PWMMR2) output is double edged else if set 0 then it is Single Edged.

2) Similarly Bits 3, 4, 5, 6 for PWM3, PWM4, PWM5, PWM6 respectively.

2) Bits 9 to 14 are used to Enable/Disable PWM outputs

1) Bit 9: If set to 1 then PWM1 output is enabled, else disabled if set to 0.

2)Similarly Bit 10,11,12,13,14 for PWM2 , PWM3 , PWM4 , PWM5 , PWM6 respectively.

8) PWMTC (PWM Timer Counter - 0xE001 4008)

The 32-bit PWM Timer Counter is incremented when the Prescale Counter reaches its terminal count. Unless it is reset before reaching its upper limit, the PWMTC will count up through the value 0xFFFF FFFF and then wrap back to the value 0x0000 0000. This event does not cause an interrupt, but a Match register can be used to detect an overflow if needed.

9) PWMPC (Prescale Counter register - 0xE001 4010)

The 32-bit PWM Prescale Counter controls division of PCLK by some constant value before it is applied to the PWM Timer Counter. This allows control of the relationship of the resolution of the timer versus the maximum time before the timer overflows. The PWM Prescale Counter is incremented on every PCLK. When it reaches the value stored in the PWM Prescale Register, the PWM Timer Counter is incremented and the PWM Prescale Counter is reset on the next PCLK. This causes the PWM TC to increment on every PCLK when PWMPR = 0, every 2 PCLKs when PWMPR = 1, etc.

3.4 Generating PWM signal

Configuring PWM, we need to enable the outputs and select PWM functions for the corresponding PIN on which output will be available. But first we need to know values of periodic time, the resolution using a prescale value and pulse widths.

Setting and Initializing the PWM device as per the following steps:

1. Select the PWM function for the PIN on which you need the PWM output using PINSEL0/1 register.
2. Select Single Edge or Double Edge Mode using PWMPCR. By default it is Single Edge Mode.
3. Assign the Calculated value to PR.
4. Set the Value for PWM Period in PWMMR0.
5. Set the Values for other Match Registers i.e. the Pulse Widths.
6. Set appropriate bit values in PWMMCR like for e.g. resetting PWMTC for PWMMR0 match and optionally generate interrupts if required.
7. Set Latch Enable Bits for the Match Registers that are used.
8. Then Enable PWM outputs using PWMPCR.
9. Reset PWM Timer using PWMTCR.
10. Finally, Enable Timer Counter and PWM Mode using PWMTCR.

3.5 Code

Hence, following the steps above and learning the PWM register and its bits, we now write a code for driving servo motor by programming LPC2148.

```
#include <lpc214x.h>

void DelaymSec(unsigned int j)
{
unsigned i;
for(; j > 0; j--)
{
for(i = 0; i < 10000; i++);
}
}

void initServoPWM()
{
PINSEL0&=0xFFFFFFFC;
PINSEL0|=0x00000002; //Enabling P0.0 as PWM1

PWMPR = 150; //PWM Prescaler PCLK/150 = 100KHz
PWMPC = 0; //PWMPC increments on every PCLK
PWMTC = 0; //PWMTC increments on every PWMPC=PWMPR
PWMMR0 = 2000; //PWM base frequency 100KHz/2000=50Hz
PWMMR1 = 60;
PWMMR2 = 0;
PWMMR3 = 0;
PWMMR4 = 0;
PWMMR5 = 0;
PWMMR6 = 0;
PWMMCR = 0x00000002;
PWMPCR = 0x2600;
PWMLER = 0x7F;
PWMTCR = 0x01;
}

void servo_1(unsigned char degrees)
{
float regval = 0;
regval = ((float)degrees / 1.125) + 60.0;
PWMMR1 = (unsigned int)regval;
PWMLER = 0x02;
}

void Servo_1_Free(void)
{
PWMMR1 = 1999;
PWMLER = 0x02;
```

```
}

void init_devices(void)
{
initServoPWM(); //Initialise PWM for servo
}

int main(void)
{
//Set all pins as GPIO
PINSEL0 = 0x00000000;
PINSEL1 = 0x00000000;
PINSEL2 = 0x00000000;

//Initialise Peripherals
init_devices();
DelaymSec(2000);
while(1)
{
Servo_1(90);
DelaymSec(600);
Servo_1(180);
DelaymSec(600);
Servo_1(90);
DelaymSec(600);
Servo_1(0);
DelaymSec(600);
}
}
```

Chapter 4

Referneces and Futher Reading

References and Further Reading:

1. International Journal of Computer Applications
www.research.ijcaonline.org/volume64/number13/pxc3885609.pdf
2. Society of Robots
http://www.societyofrobots.com/member_tutorials/book/export/html/228
http://www.societyofrobots.com/member_tutorials/node/230
3. OC Freaks
www.ocfreaks.com
4. The Technical Brain
http://thetechnicalbrain.com/electronics/elec_servo.php
5. Max Embedded
www.maxembedded.com
6. ATmega2560 Datasheet
http://www.atmel.com/images/atmel-2549-8-bit-avr-microcontroller-atmega640-1280-1281-2560-2561_datasheet.pdf
7. LPC2148 User Manual
http://www.nxp.com/documents/user_manual/UM10139.pdf
8. Fire Bird V LPC2148 ARM7 Robotic Research Platform, Hardware and Software Manual
9. Youtube
Servos; working principle and homemade types : https://www.youtube.com/watch?v=v2jpnyKPH64
How do servos work : https://www.youtube.com/watch?v=-XSXfqd1N58